FORSCHUNGSZENTRUM JÜLICH GmbH
Zentralinstitut für Angewandte Mathematik
D-52425 Jülich, Tel. (02461) 61-6402

Interner Bericht

Conjugate Gradients Parallelized on the Hypercube

Achim Basermann

KFA-ZAM-IB-9309

September 1993
(Stand 01.09.93)

Erscheint im "International Journal of Modern Physics C"

CONJUGATE GRADIENTS PARALLELIZED ON THE HYPERCUBE

ACHIM BASERMANN
Central Institute for Applied Mathematics
Research Centre Jülich GmbH, 52425 Jülich, Germany
email: acb@zam036.zam.kfa-juelich.de

ABSTRACT

For the solution of discretized ordinary or partial differential equations it is necessary to solve systems of equations with coefficient matrices of different sparsity pattern, depending on the discretization method; using the finite element method (FE) results in largely unstructured systems of equations. A frequently used iterative solver for systems of equations is the method of conjugate gradients (CG) with different preconditioners. On a multiprocessor system with distributed memory, in particular the data distribution and the communication scheme depending on the used data struture are of greatest importance for the efficient execution of this method. Here, a data distribution and a communication scheme are presented which are based on the analysis of the column indices of the non-zero matrix elements. The performance of the developed parallel CG-method was measured on the distributed-memory-system INTEL iPSC/860 of the Research Centre Jülich with systems of equations from FE-models. The parallel CG-algorithm has been shown to be well suited for both regular and irregular discretization meshes, i.e. for coefficient matrices of very different sparsity pattern.

Keywords: Conjugate gradients; Finite elements; Parallelization; Data distribution; Communication scheme.

1. Introduction

For the solution of discretized ordinary or partial differential equations it is necessary to solve systems of equations with coefficient matrices of different sparsity pattern, depending on the discretization method; using the finite element method (FE) results in largely unstructured systems of equations, using the finite difference method results in coefficient matrices of more regular pattern, e.g. banded or block matrices. A frequently used iterative solver for systems of equations is the method of conjugate gradients (CG) with different preconditioners.[1,2]

In 1990, Aykanat e.a. presented a modified CG-algorithm[3] with better parallelization properties than the original method developed by Hestenes and Stiefel. The main work in each iteration of this method consists in the computation of the matrix-vector product. Thereby, accessing the vector is determined by the sparsity pattern and the storage scheme of the matrix. On a multiprocessor system with distributed memory, in particular the data distribution and the communication scheme depending on the used data struture are of greatest importance for the efficient computation of the matrix-vector-product. Here, a data distribution and a communication scheme are presented which are based on the analysis of the column indices of the non-zero matrix elements.

First, the matrix is distributed row-wise on each processor, the vector com-

ponents accordingly. By analyzing the column indices, each processor determines which matrix elements result in computations with local data and which ones with non-local data. Communication and local computations are performed overlapped to reduce the contribution of the communication time to the total execution time. The data distribution and the communication scheme merely depend on the sparsity pattern of the matrix, not on the values of the non-zero elements. The schemes are determined once before the execution of the CG-iteration.

The performance of the developed parallel CG-method was measured on the distributed-memory-system INTEL iPSC/860 of the Research Centre Jülich with equation systems from two FE-models. The first FE-model comes from environmental science; it simulates the behaviour of pollutants in geological systems.[4,5] In the second FE-model from structural mechanics, stresses in materials induced by thermal expansion are calculated by applying the FE-program SMART.[6]

2. The Method of Conjugate Gradients

The method of conjugate gradients[1] is an algorithm for solving systems of linear equations $Ax = b$, particularly for sparse coefficient matrices A. The method converges for matrices, which are symmetric and positive definite.

Aykanat e.a.[3] suggested a modified CG-algorithm (see Algorithm 1) which has better parallelization properties than the original method.

Algorithm 1. The modified CG-method

Choose an arbitrary $x_0 \in \mathbb{R}^n$;

$$\begin{aligned} g_0 &= Ax_0 - b \\ d_0 &= -g_0 \end{aligned}$$

$i = 0, 1, \ldots$

$$\begin{aligned} \alpha_i &= \frac{g_i^T g_i}{d_i^T A d_i} \\ \beta_i &= \frac{\alpha_i (Ad_i)^T A d_i}{d_i^T A d_i} - 1 \\ g_{i+1}^T g_{i+1} &= \beta_i g_i^T g_i \\ x_{i+1} &= x_i + \alpha_i d_i \\ g_{i+1} &= g_i + \alpha_i A d_i \\ d_{i+1} &= -g_{i+1} + \beta_i d_i \end{aligned}$$

until $\|g_{i+1}\|_2 \leq \epsilon_r$.

In each iteration, the vectors x_i, g_i, and d_i are computed. x_i approximates the solution vector, g_i is the residue; d_i determines the direction in which the next approximation of the solution vector is searched for. The main work in each iteration consists in the computation of the matrix-vector-product Ad_i. Furthermore, two dot products and three vector additions have to be performed. Iteration is continued until the euclidean norm of the residue is less than or equal to ϵ_r. Another stopping criterion which uses the maximum scaled absolute difference of the components of the latest two approximations of the solution vector is determined as follows:

$$\max_{j=1,\ldots,n} 2\frac{|x_{i+1}^j - x_i^j|}{|x_{i+1}^j| + |x_i^j|} \leq \epsilon_s . \tag{2.1}$$

The main difference between the modified and the original CG-method is that in Algorithm 1 the constants α_i and β_i and therefore all dot products are computed at the beginning of each iteration. If each iteration is performed in parallel on a distributed-memory-system the local values of the dot products can be included in one message for determining the global values.[2]

3. Storage Scheme

Storage schemes for large sparse matrices depend on the sparsity pattern of the matrix, the considered algorithm, and the architecture of the used computer system. In literature, many variants of storage schemes can be found in e.g. (Refs. 7-12).

The storage scheme considered here is often used in FE-programs and suitable for regular as well as for irregular discretization meshes. It can be found in similar form in e.g. (Ref. 7). The principle of the scheme is shown in (3.3) for matrix (3.2).

The non-zeros of matrix (3.2) are stored row-wise in three one-dimensional arrays. a^w contains the values of the non-zeros, a^s the corresponding column indices. In a^z, the position of the beginning of each row in a^w and a^s is stored. The subdivisions in a^w and a^s have been added to mark the beginning of a new row. The order of the matrix elements per row in a^w and a^s is different from that in matrix (3.2) since this is ususally the case in FE-programs caused by the assembly of the coefficient matrix from the single elements.

$$A = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 9 & 0 & 0 & 0 & 0 & 0 \\ 0 & 9 & 3 & 10 & 0 & 0 & 0 & 0 \\ 0 & 0 & 10 & 4 & 11 & 14 & 12 & 18 \\ 0 & 0 & 0 & 11 & 5 & 0 & 17 & 0 \\ 0 & 0 & 0 & 14 & 0 & 6 & 15 & 0 \\ 0 & 0 & 0 & 12 & 17 & 15 & 7 & 0 \\ 0 & 0 & 0 & 18 & 0 & 0 & 0 & 8 \end{pmatrix} . \tag{3.2}$$

$$
\begin{aligned}
a^w &= (1 \mid 9\ 2 \mid 10\ 9\ 3 \mid 10\ 4\ 18\ 14\ 12\ 11 \mid 11\ 5\ 17 \mid 15\ 14\ 6 \mid 12\ 17\ 7\ 15 \mid 18\ 8), \\
a^s &= (1 \mid 3\ 2 \mid 4\ 2\ 3 \mid 3\ 4\ 8\ 6\ 7\ 5 \mid 4\ 5\ 7 \mid 7\ 4\ 6 \mid 4\ 5\ 7\ 6 \mid 4\ 8), \\
a^z &= (1\ 2\ 4\ 7\ 13\ 16\ 19\ 23\ 25). \qquad (3.3)
\end{aligned}
$$

4. Parallelization

4.1. Data Distribution

For parallelizing Algorithm 1 on a distributed-memory-system, the matrix and vector arrays must be suitably distributed to each processor. For the considered data distribution schemes, the arrays a^w and a^s are distributed row-wise; the rows of each processor succeed one another. The distribution of the vector arrays corresponds component-wise to the row distribution of the matrix arrays.

Criteria for the data distribution can be: each processor gets the same number of rows or so many rows that each processor has nearly the same number of non-zeros. The number of arithmetical operations for the computation of the matrix-vector-product is proportional to the number of non-zeros; the remaining arithmetical operations of one iteration are proportional to the number of rows. The criterion considered here is that each processor has to compute nearly the same number of arithmetical operations. If the discretization mesh is regular, i.e. the sparsity pattern of the coefficient matrix is regular, all three criteria result in nearly the same data distribution. If the mesh is very irregular, the three distributions are considerably different.

In the latter case, processor k, $k = 0, \ldots, p-1$, gets so many rows until criterion

$$
\frac{e_k + \xi n_k}{e + \xi n} \geq \frac{1}{p}, \qquad \text{for } e_k, n_k \gg 10 \qquad (4.4)
$$

is satisfied for the first time, i.e. for the least number of rows possible. p is the number of the used processors, e_k the number of non-zeros, and n_k the number of rows of processor k. e is the total number of non-zeros and n the order of the matrix. The parameter ξ considers the number of vector-operations besides the operations of the matrix-vector-product and the ratio of the execution times of multiplication, division etc. operations and the addition operation; it is therefore dependent on the processor architecture. For the INTEL i860XR, ξ is about 13. The numerator in (4.4) is proportional to the number of arithmetical operations of one iteration of processor k, the denominator is proportional to the total number of arithmetical operations of one iteration.

With these considerations, the contribution of the matrix-vector-product to one iteration can be approximated by

$$
a_{\mathrm{MVP}} \approx \frac{e}{e + \xi n} = \frac{1}{1 + \xi / m_z}, \qquad \text{for } e, n \gg 10. \qquad (4.5)
$$

$m_z = e/n$ is the mean number of non-zeros per row.

The data distribution according to criterion (4.4) is shown in (4.6) by distributing matrix (3.2) to four processors. The other arrays are distributed analogously.

$$\begin{aligned}
&\text{Processor 0: } a_0^w &=& \ (1 \mid 9\ 2 \mid 10\ 9\ 3),\\
&\text{Processor 1: } a_1^w &=& \ (10\ 4\ 18\ 14\ 12\ 11 \mid 11\ 5\ 17),\\
&\text{Processor 2: } a_2^w &=& \ (15\ 14\ 6 \mid 12\ 17\ 7\ 15),\\
&\text{Processor 3: } a_3^w &=& \ (18\ 8).
\end{aligned} \tag{4.6}$$

4.2. Communication Scheme

On a distributed-memory-system, the computation of the matrix-vector-product requires communication because each processor possesses only a part of the components of the vector. For the efficient computation of the matrix-vector-product, it is necessary to develop a suitable communication scheme.

First, the arrays a_k^s are analysed on each processor k to determine which data result in accesses to components of d_i of other processors. Then, a_k^s and a_k^w are rearranged in such a way that the data which result in accesses to processor h are collected in block h. The data of block h succeed row-wise one another with increasing column index per row. Block k is the first block in a_k^s and a_k^w and contains the data which result in local accesses. The goal of the rearranging is performing computation and communication overlapped.

The principle of the rearranging is shown in (4.7) for the data distribution from (4.6). Here, merely array a_1^s is analysed and rearranged.

Computing the operation row times vector of the matrix-vector-product of processor 1, the index 3 results in an access to component d_i^3 of processor 0, the index 8 to d_i^8 of processor 3, and the indices 6 and 7 in accesses to d_i^6 and d_i^7 of processor 2. The data blocks in (4.7) are separated by double dashes for elucidation; the blocks have been numbered below the brackets. After rearranging, the data of block 1 result in local accesses, the data of block 0 in accesses to processor 0, the data of block 2 in accesses to processor 2, and the data of block 3 in accesses to processor 3.

$$\begin{aligned}
&\text{Processor 0: } a_0^s &=& \ (1 \mid 3\ 2 \mid 4\ 2\ 3), \quad d_{i,0} = (d_i^1\ d_i^2\ d_i^3)\\
&\underline{\text{Processor 1:}}\ a_1^s &=& \ (\boxed{3}\ 4\ \boxed{8}\ \boxed{6}\ \boxed{7}\ 5 \mid 4\ 5\ \boxed{7}), \quad d_{i,1} = (d_i^4\ d_i^5)\\
&\text{Processor 2: } a_2^s &=& \ (7\ 4\ 6 \mid 4\ 5\ 7\ 6), \quad d_{i,2} = (d_i^6\ d_i^7)\\
&\text{Processor 3: } a_3^s &=& \ (4\ 8), \quad d_{i,3} = (d_i^8)
\end{aligned}$$

$$\underline{\text{Rearranging:}}\ a_1^s = (\underbrace{4\ 5 \mid 4\ 5}_{1} \parallel \underbrace{\boxed{3}}_{0} \parallel \underbrace{\boxed{6}\ \boxed{7} \mid \boxed{7}}_{2} \parallel \underbrace{\boxed{8}}_{3}) \tag{4.7}$$

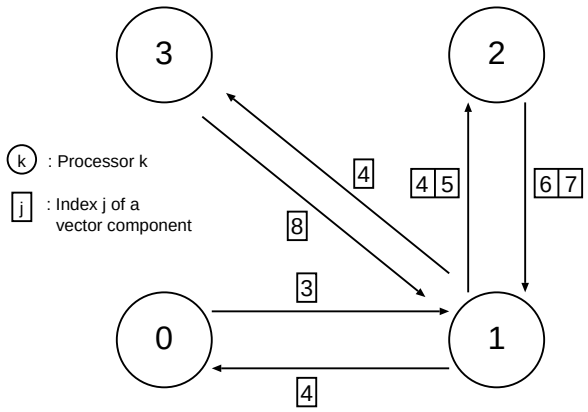

Fig. 1. Communication scheme

After having analysed the column index array a_k^s, each processor k knows which components of d_i must be required of which processors. This information is broadcasted to all processors. Then, each processor can decide which data must be sent to which processors. This communication scheme is determined once before starting the parallel CG-algorithm and applies unchanged to each iteration.

The communication scheme for the example discussed before is shown in Fig. 1.

Processor 1 e.g. receives the third component of d_i from processor 0, the sixth and seventh component from processor 2 and the eighth component from processor 3. On the other side, the fourth component of processor 1 is sent to processor 0, the fourth and fifth to processor 2 and the fourth to processor 3.

In Fig. 2, the parallel computation of the matrix-vector-product is described.

First, on each processor, the data which are necessary for other processors are sent asynchronously. After executing asynchronous receive-routines for receiving non-local data, all local computations are performed, in particular the local part of the matrix-vector-product. Then each processor waits until the data of an arbitrary processor arrive and continues the computation of the matrix-vector-product. Thereafter, each processor waits for the data of other processors until the computation of the matrix-vector-product is complete. Computation and communication are performed overlapped. While required data are on the network, operations with local or already arrived data of other processors are executed.

5. Results

The performance of the developed parallel CG-method was measured on the distributed-memory-system INTEL iPSC/860 of the Research Centre Jülich. The computer system has 32 processors with 16 Megabyte private memory each; the interconnection network has hypercube-topology. The maximum transfer rate is 2.8

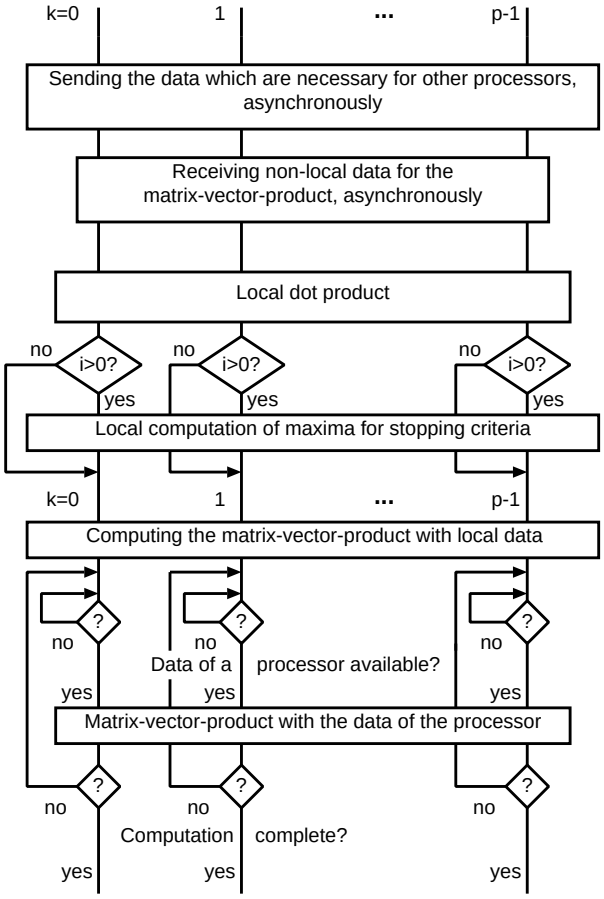

Fig. 2. The parallel matrix-vector-product

Megabyte/second per channel in both directions.

The tests presented here were performed with one equation system each of the FE-models from environmental science and structural mechanics. In Table 1, numerical data of the coefficient matrices and for the convergence of the CG-method are indicated.

The equation system from environmental science has 49392 unknows, that from structural mechanics 25222. In the first case, the mean number of non-zeros per row is near the maximum number. This is caused by a regular discretization mesh. For the second case, the mean and the maximum number are considerably different; the discritization mesh is much more irregular. Below in Table 1, the number of iterations with and without diagonal scaling, a simple preconditioner,[2] is given. The iteration was stopped when the maximum scaled absolute difference from (2.1) was

less than or equal to 10^{-5}; this corresponds to a precision of the solution vector of about five decimals. With diagonal scaling, the number of iterations is considerably smaller in both cases. The contribution of this preconditioner to the total execution time is in both cases below 1%. For the preconditioned method, the euclidean norm of the residue after 84 and 658 iterations, respectively, is given additionally.

Table 1. Numerical data of the considered equation systems

	Environmental science	Structural mechanics
Rows	49392	25222
Density	0.05%	0.6%
Non-zeros per row, max.	27	485
m_z	25.2	152.9
a_{MVP}	66%	92%
Stopping criterion: max. scal. abs. diff. $\leq 10^{-5}$		
Iterates without scaling	390	1444
Iterates with scaling	84	658
$\|g_{i+1}\|_2$	4.5×10^{-4}	1.5×10^{-5}

The sparsity patterns of both matrices are shown in Figs. 3 and 4, respectively.

The matrix from environmental science has essentially band structure with a maximum bandwidth of 2375. The matrix from structural mechanics has a much more irregular structure; the maximum bandwidth is 3474.

In Fig. 5, execution times per iteration on 32 processors with and without communication and computation performed overlapped are presented. The overlapped execution reduces the execution times by nearly 20%.

In Fig. 6, speedups on 4 to 32 processors are shown. The equation system from environmental science together with the program code and the remaining data requires the memory of more than two processors, that from structural mechanics the memory of more than four processors. For up to four and, in the second case, up to eight processors, linear speedup was assumed because nearly linear speedup was observed in tests with smaller equation systems for up to 8 processors.

For 16 processors, the speedup is 13.2 in the first case and 15.2 in the second case. This corresponds to efficiencies of 83% and 95%. For 32 processors, speedups of 21.6 and 27.2 are achieved. The efficiencies decrease to 68% and 85% because the communication overhead increases.

6. Conclusions

The developed sequential and parallel CG-algorithms have been shown to be well suited for the considered FE-models. They are used in both projects. On a distributed-memory-system, the developed data distribution and communication scheme together with the overlapped execution of communication and local compu-

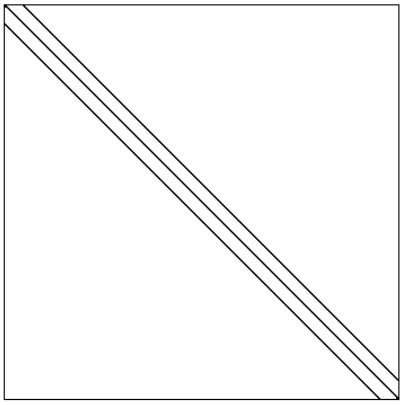

Fig. 3. Sparsity pattern of the matrix from environmental science

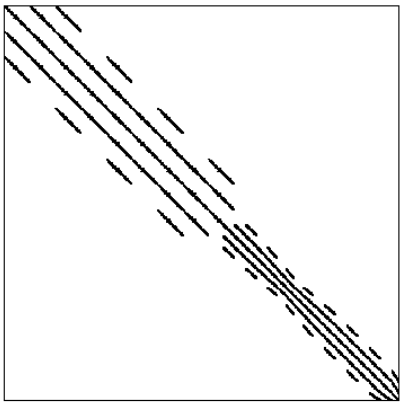

Fig. 4. Sparsity pattern of the matrix from structural mechanics

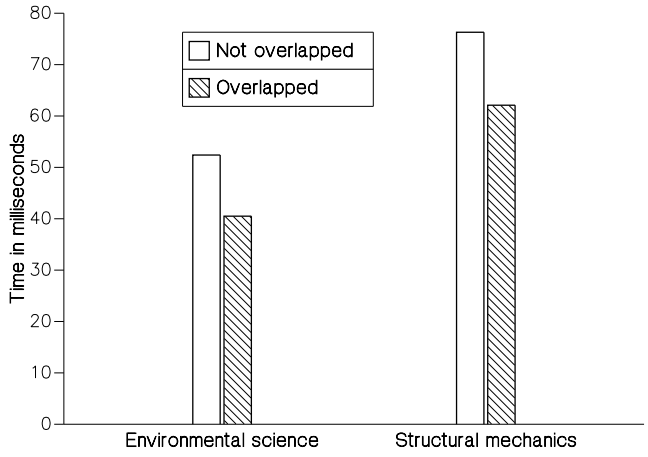

Fig. 5. Execution times per iteration

tations result in a flexible algorithm. This algorithm performs well for both regular and irregular discretization meshes, i.e. for coefficient matrices of very different sparsity patterns.

In future work other data distributions and other storage schemes for sparse coefficient matrices and their influence on the communication scheme will be tested. Moreover, other preconditioners for CG-methods[2,13,14] will be implemented. The goal of this investigation is to speed up the convergence of the CG-method and thereby to reduce the total execution time.

Furthermore, it remains to examine if the described principles of the data distribution, the communication scheme and the overlapped execution apply successfully to other algorithms, such as the QMR-algorithm for solving non-hermitian linear systems of equations[15] or Lanczos-methods for solving eigenproblems. Lanczos-methods are suited for determining eigenvalues and eigenvectors of dense and sparse matrices. In these methods, matrix-vector-products are computed and, depending on the special variant, linear systems of equations are solved. The last step is usually determining the eigenvalues of a tridiagonal matrix. A parallel algorithm for determining the eigenvalues of large real symmetric tridiagonal matrices has been already developed.[16] Large eigenproblems occur in FE-models of mechanics, in particular dynamics and quantum mechanics, and in chemistry, especially in molecular dynamics and quantum chemistry.

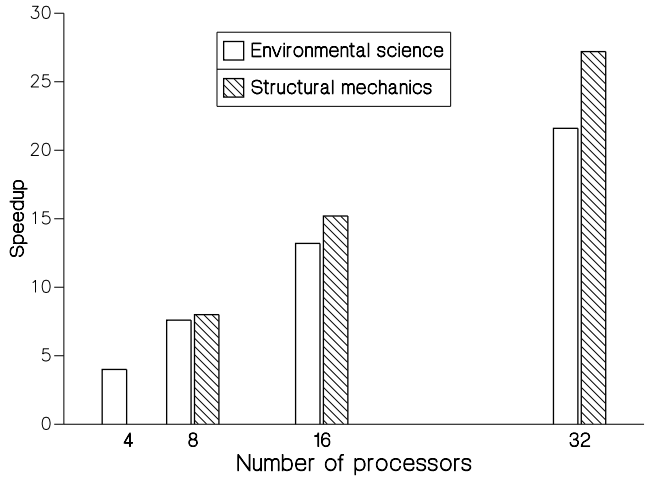

Fig. 6. Speedups

References

1. M.R. Hestenes and E. Stiefel, *Journal of Research of the National Bureau of Standards* **49** (1952), pp. 409–436.
2. J.M. Ortega, *Introduction to Parallel and Vector Solution of Linear Systems*, (Plenum Press, New York London, 1988).
3. C. Aykanat, F. Özgüner, and D.S. Scott, *Microprocessing and Microprogramming* **29** (1990), pp. 67–82.
4. G. T. Yeh, "3DFEMWATER: A Three-Dimensional Finite Element Model of Water Flow Through Saturated-Unsaturated Media", Oak Ridge National Laboratory ORNL-6386, Aug. 1987.
5. H. Vereecken, G. Lindenmayr, A.Kuhr, D.H. Welte, and A. Basermann, "Numerical Modelling of Field Scale Transport in Heterogeneous Variably Saturated Porous Media", KFA/ICG-4 Internal Report No. 500393, Jan. 1993.
6. "SMART, Benutzerhandbücher", Univ. Stuttgart ISD-Berichte, 1976–1992.
7. C.P. Kruskal, L. Rudolph, and M. Snir, *Theoretical Computer Science* **64** (1989), pp. 135–157.
8. S. Pissanetsky, *Sparse Matrix Technology*, (Academic Press, London Orlando, 1984).
9. P. Fernandes and P. Girdinio, *Parallel Computing* **12** (1989), pp. 327–333.
10. O.A. McBryan and E.F. Van de Velde, *Parallel Computing* **5** (1987), pp. 117–125.
11. U. Schendel, *Sparse–Matrizen*, (R. Oldenbourg Verlag, München Wien, 1977).
12. N. Feistl, "Das Verfahren der konjugierten Gradienten auf Vektorrechern", Ludwig-Maximilians-Universität München, Diplomarbeit, July 1987.

13. S.F. Ashby, *SIAM J. Matrix Anal. Appl.* **12** (1991), pp. 766–789.
14. G. Pini and G. Gambolati, *Adv. Water Resources* **13** (1990), pp. 147–153.
15. R.W. Freund and N.M. Nachtigal, *Numerische Mathematik* **60** (1991), pp. 315–339.
16. A. Basermann and P. Weidner, *Parallel Computing* **18** (1992), pp. 1129–1141.